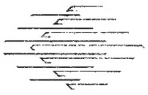

Carnegie Mellon
Software Engineering Institute

Mining Existing Assets for Software Product Lines

John Bergey
Liam O'Brien
Dennis Smith

May 2000

Product Line Practice Initiative

Unlimited distribution subject to the copyright

Technical Note
CMU/SEI-2000-TN-008

DTIC QUALITY INSPECTED 4

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2000 by Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Contents

1	**Introduction**	**1**
2	**Product Lines and Mining of Assets**	**3**
3	**Information Gathering and Decision-Making**	**5**
	3.1 Preliminary Information Gathering: General Baseline Data	5
	3.2 Decision Making: Options Analysis for Reengineering (OAR)	6
4	**Activities in Mining of Assets**	**10**
	4.1 Technical Understanding of Assets	10
	4.2 Rehabilitation of Software Assets	11
5	**Architecture Reconstruction**	**13**
	5.1 Role of Reconstruction	13
	5.2 Dali Workbench	14
	5.3 Related Architecture Reconstruction Efforts	16
6	**Summary and Conclusions**	**18**
References		**19**
Appendix: Definition of Key Concepts		**21**

List of Figures

Figure 1: Horseshoe Model for Integrating Reengineering and Software Architecture 6
Figure 2: Overview of the Dali Workbench and the Various Phases 15

Abstract

Mining of existing assets offers an organization the potential to leverage all, or part, of its cumulative system investments, and thus represents a critical practice area in implementing a software product line. However, there are significant risks in achieving success because of the poorly documented and maintained state of many existing systems and the fact that many systems were initially developed for different paradigms than current distributed, Web-oriented, object-oriented approaches.

Four basic steps are required to successfully mine assets: 1) preliminary information gathering, 2) making decisions on whether to mine assets and which type of overall strategy to use, 3) obtaining detailed technical understanding of existing software assets, and 4) rehabilitation of assets.

This note outlines basic considerations for each of these steps. It outlines typical information to collect before an analysis. It then outlines a model for making decisions on mining legacy assets, and discusses the technical understanding of assets and the rehabilitation of assets.

Because of its importance as a strategy for product lines, architecture reconstruction is discussed, as it is supported by an automated tool set known as the Dali workbench.

1 Introduction

Few systems, whether individual systems or product lines, start out as "green field" development efforts. Instead, applications are usually built as extensions of legacy systems. Often the term "mining existing assets" simply refers to finding useful legacy software from an organization's existing inventory of software applications, and reengineering it to fit within a new application. However, current best reengineering practice suggests that such a view misses the big picture behind software evolution and mining of existing assets. For example, in a recent review of reverse engineering, Muller suggests a need to focus reverse engineering at the more significant levels of the software architecture and the business processes as a precursor to understanding how, and if, existing assets can be leveraged [Muller 00].

Two situations where software understanding at higher levels of abstraction is especially important include migration to a modern architecture and the development of a product line. In some cases, it will still be necessary to mine assets at lower levels of abstraction, but at a minimum it is necessary to understand the architectural and functional features of the system.

Although many legacy systems do not have up to date documentation and other artifacts, when these are available they can serve as potential candidates for mining. Relevant assets can include architecture descriptions, domain models, design and usage documentation, test programs, test data and documentation, interface specifications, tools, code, and processes. In addition it is important to understand architecture and design tradeoffs, engineering constraints, and application domain knowledge.

Overall, four basic steps are required to successfully mine assets: 1) preliminary information gathering, 2) making decisions on whether to mine assets and which type of overall strategy to use, 3) performing analyses to obtain a detailed technical understanding of existing software components and their relationships and interfaces, and 4) carrying out the rehabilitation of selected assets. These overall steps are outlined to provide a starting point for organizations considering the mining of assets. References are provided for more detailed information. Before describing each of these steps (Sections 3 and 4), Section 2 introduces salient issues and factors to consider when mining assets for product lines. Section 3 outlines preliminary information to gather. It then discusses decision making for mining of assets and outlines a model for making decisions on technical strategies. Section 4 outlines the major activities that are required for the technical understanding of assets at the component level, their relationships and interfaces. It then discusses the rehabilitation of assets.

Architecture reconstruction is a technical strategy that is particularly relevant to enabling the mining of assets for product lines. Because of its importance, Section 5 provides an overview of architecture reconstruction as supported by an automated tool set, the Dali workbench. Section 6 provides a summary and conclusions.

Since this note focuses on mining existing assets for product lines, the Appendix provides definitions of our use of the term product line and other key architecture and product line concepts.

2 Product Lines and Mining of Assets

Product lines are based on a software architecture, which describes and captures key "change" or predicted variability points for future evolution, and which forms the backbone for building software intensive systems. The architecture represents the earliest set of design decisions for a system, and thus represents an irreversible foundation for future developments of the system. Quality needs to be built in at the software architecture level and it can't be appended at implementation time. Individual products within a product line will need to exhibit variability in function, and the capability to be re-targeted to different functional goals. As a result it is even more important to have an appropriate software architecture for a software product line than for individual systems.

While mining assets can often provide a cost effective means of leveraging an organization's existing system capabilities, mined assets must have properties that are consistent with the corporate drivers of the product line architecture. The assets need to fit into an architecture that will be long-lived and that is designed to satisfy carefully developed functional goals and well thought out non-functional quality attributes.

Moreover, product lines focus on strategic, large-grained reuse of the mined assets. Because the mining of assets is resource intensive, the most desirable assets for mining are those that make up large patterns of interoperation in the legacy architecture and that clearly satisfy specific requirements in the new product line architecture. Thus, reuse is not restricted to single components, but rather to entire assemblies of components and their pre-defined and supported interactions or patterns. However, in some cases, individual components can be mined as core assets if they fit cleanly within the product line architecture and offer significant leverage across the products in the product line.

The primary drivers that motivate large-scale reuse for a product line are schedule, cost and quality. An initial rough estimate of the cost (and schedule) of carrying out mining should be developed to determine whether to go down the road of mining at all. In some cases mining of assets may not be practical or worth carrying out.

The inclusion of extracted software components or assemblies of components is economically feasible when a project that uses those components can be completed at lower cost, meet architectural quality requirements, and produce equal or greater functionality than creating similar assets from scratch. Any calculation of reuse cost should include the total cost of asset use over the lifetime of the product or products, and not just the cost of mining/restoring a particular set of assets. In practice, improvements on just one of the scales of schedule, cost and time may produce a significant tactical advantage. For example, if mining and restoration gain time, but result in slightly reduced functionality (relative to

building from scratch), the case for mining could be a strong one if time to market were a primary driver for the effort.

The qualities that make assets desirable are different for product line usage as opposed to single system usage. When mining assets for single systems, it is critical that the assets perform specific functions very well. For product lines it is important for the asset base to be able to accommodate extensive variability in function, while still being able to fulfill quality attributes. Quality attributes such as maintainability and portability may assume even greater importance in a product line because these attributes affect the asset's total cost of ownership across a family of products. For example, a bank may have a batch application that calculates the projected value of an investment account given certain dates and portfolio allocations. Porting the code to a new family of interactive systems requires consideration of performance and security issues and the architectural "fit" with the new interactive system in addition to the functional logic.

In analyzing an asset's potential role for use in a product line, factors to consider include

- its usefulness for immediate products (i.e., as start-up assets)
- its application for potential future products (i.e., as long-lived assets)
- the amount of effort required to make the component's interface compatible with the product line architecture
- the potential (and nature) of future changes to the asset based on anticipated evolution requirements for the product line architecture

3 Information Gathering and Decision-Making

Given the constraints that the software architecture places on potential product line assets, it is important to have a disciplined approach to decision making. The first step in making informed decisions is to gather preliminary information to create a baseline of knowledge about the characteristics of the product line and the organization's current assets. This step will help to create an understanding of where mined assets may fit into a product line architecture and which types of software assets represent viable candidates for mining. Once the baseline is established, decisions on basic strategies for mining assets can be made.

Section 3.1 outlines the general baseline data that needs to be gathered prior to the effort of mining assets. Section 3.2 discusses decision making and mining strategies from the perspective of Options Analysis for Reengineering (OAR), a model to guide decisions on overall strategies for mining assets.

3.1 Preliminary Information Gathering: General Baseline Data

A first step in the mining of assets involves establishing a baseline to provide the general background information for making decisions on mining assets. This information includes the following:

- description of the product line architecture, the scope of the product line, and potential changes
- experience of the organization with mining and reengineering
- catalog of available corporate and legacy system assets and documentation for the assets
- needs of the product line for potential mined assets (start-up and long term)
- identification of potential options for fitting mined assets into the product line architecture
- a preliminary estimate (to be verified later during a more complete analysis) of which assets are mining candidates requiring black box changes (primarily changes to interfaces) and which are candidates requiring white box changes (significant changes to underlying software)
- maintenance history of the assets under consideration
- resources available for the mining activity
- availability of the people who currently maintain the assets of interest
- preliminary feasibility estimates (to be verified subsequently with more detailed analysis) of cost, functionality, and quality attribute tradeoffs for mining versus building from scratch

3.2 Decision Making: Options Analysis for Reengineering (OAR)

Once the baseline data has been collected, a set of assets may emerge as potential candidates for mining for a product line. Mining of assets is a complex activity with many potential paths. Options Analysis for Reengineering (OAR) [Bergey 99] enables practitioners to determine the basic strategies and technical options that are applicable for different types of problems.

OAR provides help in making two types of fundamental decisions when mining assets. These are the following:

1. determining the level of analysis that is appropriate to a problem, such as the code, function, and architecture levels. In most cases when mining assets for product lines, the architectural level will need to be considered, at least to the extent of determining appropriate interfaces; however, it is possible that analysis at the code and function level will also be required
2. determining the particular reengineering strategy that can best exploit the mining of existing assets

The "Horseshoe" Model

OAR uses the visual metaphor of a "horseshoe" to establish the context to address both of these types of decisions. As shown in Figure 1, this horseshoe model that is described by Woods integrates the code-level and architectural reengineering views of a software system [Woods 99].

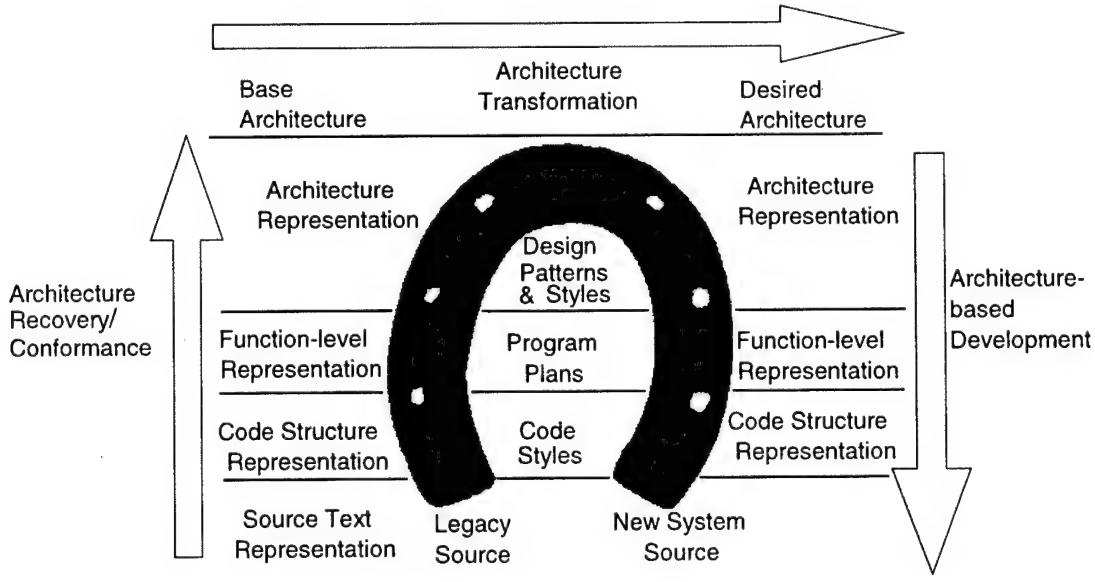

Figure 1: Horseshoe Model for Integrating Reengineering and Software Architecture

Levels of Abstraction

The horseshoe model can first be used to help in understanding and making decisions on the appropriate level of abstraction. The model represents data at the following three levels:

- "Code-structure representation" includes source code and artifacts such as abstract syntax trees (ASTs) and flow graphs obtained through parsing and rote analytical operations. At the code-structure level, there are actually two sub-levels, represented in Figure 1. These are a) source text (or string-based) transformations and b) code structure transformations, such as those based on the abstract syntax tree.
- "Function-level representation" describes the relationship between the programs functions (calls, for example), data (function and data relationships), and files (groupings of functions and data).
- "Concept" level represents clusters of both function and code level artifacts that are assembled into subsystems of related architectural components or concepts.

Choice of Appropriate Strategy for Mining Assets

As we pointed out in Section 2, the most desirable assets for mining for product lines are represented by large assemblies of components and their pre-defined interactions and patterns. In many cases, though, individual components can be mined as core assets if they offer a clean fit within the product line architecture. When mining large assemblies of components, an organization will focus on understanding (or reconstructing) the architectural structure and interactions of the components. Types of strategies that support different types of goals for mining these kinds of assets are outlined in the next section, "Mining Large Grained Assets at the Architectural Level." When mining smaller components, the focus will be on understanding at the program level, and the types of strategies that are outlined in "Mining Smaller Grained Components" on page 8 will be more appropriate.

Mining Large Grained Assets at the Architectural Level

When large grained assets are mined at the architectural level, they are interpreted at a high level of abstraction with careful consideration of component relationships and interactions in order to maximize leverage in an evolved overall compositional structure. For this type of problem, three basic strategies (or combination of these strategies) can be selected, depending on the goals of the organization. These strategies, which are represented along the outside of Figure 1, are the following:

1. *Architecture recovery and conformance* goes up the left leg of the horseshoe. It is used when the original architecture has been eroded and there is a need to reconstruct the architecture of the existing system (e.g., use it as a baseline for a product line architecture). Architecture reconstruction extracts artifacts from the code, and constructs an extracted view of the system, consisting of a set of components and the relationships between them. The reconstructed architecture can be evaluated for its conformance to the as-designed architecture. It can also be evaluated with respect to a number of quality attributes such as performance, modifiability, security, or reliability.

Because of its importance for mining assets for product lines, architecture reconstruction and conformance is examined in more detail in Section 5.

2. *Architecture transformation* goes across the top of the horseshoe. It is used when the goal is to migrate from an as-built (or an as-reconstructed) architecture to a desirable new architecture, such as when a current architecture doesn't fulfill certain quality attributes that are needed to meet goals of one or more products in a product line. The desired architecture is re-evaluated against the system's quality goals and subject to other programmatic and economic constraints.
3. *Architecture-based development* (ABD) [Bass 99] goes down the right side of the horseshoe to instantiate the desired architecture. This approach is used when a new or updated product line architecture is needed. The legacy assets can represent either a starting point or a set of interconnected components that will fit into the new architecture. Designing an architecture for a product line or long-lived system is difficult because detailed requirements are not known in advance. The ABD method fulfills functional, quality, and business requirements at a level of abstraction that allows for the necessary variation when producing specific products.

Mining Smaller Grained Components

Although the mining of assets for product line systems will usually focus on large grained assets, there may be times when the mining of smaller grained assets is appropriate for insertion into the product line architecture. In these cases, shortcuts across the horseshoe, or combinations of these paths, may be appropriate. These "shortcut" paths can represent pragmatic choices based on organizational or technological constraints, such as the availability of reengineering tools. They could also represent a realistic response to a defined goal.

Shortcut paths across the horseshoe include the following:

1. code level changes, where assets are identified and changed only at the code level to meet system needs. In addressing the Y2K problem, there was often not a need to change the underlying structure of the system or to add functionality. The most logical choice to address Y2K issues often involved simple manipulations of the source code.
2. function level changes, where assets are adapted for different functional requirements. These changes often involve "white box" changes where the actual code structure of assets are changed to fit into a different design structure.
3. architectural level changes, where assets retain some of the initial core functionality but need to have new types of interactions and fit new architectural patterns. In these cases, wrapping approaches are often used where the core functionality is retained and the interfaces are changed. For example, moving an application to a Web-based approach can often use self-contained existing components that have been "wrapped" to fit within the new architectural constraints of the Web. Several of these approaches are outlined in Section 4.2.

In practice, many actual applications will use a combination of several paths, either going along the outside of the horseshoe, or across the horseshoe. In these cases the horseshoe

represents a useful model for making conscious decisions on the appropriate combination of paths to consider.

4 Activities in Mining of Assets

The OAR approach outlined in Section 3 will guide an organization in deciding on an overall strategy. Once decisions on a particular strategy are made, the detailed work of mining of assets is undertaken. Although the specific tasks and activities will differ depending on the overall strategy and a lower level instantiated process for fulfilling the strategy, the detailed technical work can be broken down into

1) technical understanding of software assets

2) rehabilitation of assets

Technical understanding of software assets, outlined in Section 4.1, provides an in-depth analysis of the software components, relationships and interfaces of the existing system. Rehabilitation of assets, outlined in Section 4.2, makes changes to existing assets to enable them to be of value to the product line. Emerging techniques for asset rehabilitation that involve wrapping approaches are outlined in "Software Asset Rehabilitation Approaches" on page 12.

4.1 Technical Understanding of Assets

Tilley [Tilley 98] lists three major activities in the technical understanding of assets.[1] These activities represent ideal types that are always followed but where there are strong variations in the level of detail required. For example, when architecture reconstruction is required, the activities of technical understanding are detailed and time-consuming because every software component and relationship in the system needs to be analyzed. On the other hand, when assets are to be wrapped for insertion into a product line architecture, a much lower level of understanding is required because the primary focus is on the component interfaces.

The major activities in technical understanding of assets are the following:

1. *Detailed data collection* gathers comprehensive information about the system and its software, components, and related artifacts. As pointed out above, the level of detail to be collected will vary depending on the strategy chosen. This activity often uses a combination of computer-aided tools and techniques, together with corporate knowledge and experience. The raw data from the existing system are used to identify a system's software artifacts and relationships. This software data collection activity is an essential first step for constructing and exploring higher-level abstractions. Techniques for data collection include system examination, document scanning, and experience

[1] We start with Tilley's categorization and make minor modifications to fit the specific needs of mining assets for product lines.

capture. Sources of data gathering include compiler-based static analysis, natural language content analysis (from documentation and source code comments), and interviewing.

2. *Knowledge management* is the capturing, organizing, understanding, and extending of past experiences, processes and individual know-how. Knowledge management is most typically approached by organizing the data in a model, such as a domain model, relational model, or an object model.

3. *Exploration* involves analyzing the meaning of the captured data through a set of activities that includes navigation, analysis, and presentation. This activity of exploration is the key to increased comprehension because it facilitates the iterative refinement of hypotheses. The presentation of analysis results has traditionally taken the form of charts, tables, or graphs. Multimedia-enhanced computers introduce new ways of presenting this information. Flexible architecture visualization techniques and multiple views enable the engineer to recognize and appreciate patterns and motifs such as central, fringe, and isolated software components.

4.2 Rehabilitation of Software Assets

Once the software assets are understood at a sufficient level of detail, rehabilitation takes place. Large grained assets at the architectural level may require rehabilitation through the strategy of architecture transformation (for example, to fulfill different quality attributes). They may also represent a starting point for inserting a set of interconnected components into a new architecture developed through the strategy of architecture based development. Large grained software assets that are well structured, well documented and have been used effectively over long periods of time can sometimes be applied as core product line assets with little or no change.

Smaller grained assets that will require only the interfaces to be changed, rather than large chunks of the underlying code or algorithms, can be wrapped to satisfy new interoperability requirements. In these cases, the tasks of technical understanding and rehabilitation will be relatively straightforward. On the other hand, software assets that don't satisfy these properties will require more detailed technical understanding and rehabilitation, and they may have higher maintenance costs over the long term.

Usually, an asset will need to be changed to accommodate the constraints of the architecture. However, it is worth examining the effort required to make a minor change to the architecture. If this is done, it is critical to understand the implications of the architecture change for other components, and for future projected changes.

Mined assets should not dictate the design or architecture of a new product development effort. Rather, they should be viewed as making up a resource pool that can be used in product development but which present unique opportunities and risks.

Software Asset Rehabilitation Approaches

Comella-Dorda recently identified a set of rehabilitation approaches based on different forms of wrapping [Comella-Dorda 00]. Depending on the type of software asset and its potential use, a subset of these approaches can often be appropriate for rehabilitating assets. These approaches are most relevant when existing software components are mined for inclusion in a product line architecture. The set of approaches includes the following:

- *User interface modernization* – makes the user interface of an asset more usable. One common technique is screen scraping, which wraps an old, text-based interface with a new graphical interface.
- *Data modernization*, or *data wrapping* – enables the accessing of legacy data through different interfaces or protocols than those for which the data was initially designed. Examples of data modernization include
  a. database gateway, which translates between two or more data access protocols, thus permitting components to access databases using standard protocols (such as Sun's Java Database Connectivity (JDBC) or Microsoft's Open Database Connectivity (ODBC)), rather than vendor specific protocols
  b. XML™ integration which uses the Extensible Markup Language (XML) for data integration between applications; this solution is especially relevant for the automated exchange of information between systems from different organizations
- *Functional or logic modernization* – encapsulates not only the legacy data, but also the business logic of a legacy system. Examples includes object-oriented wrapping, which translates the monolithic semantics of a procedural system to the richly structured semantics of an object-oriented system.

5 Architecture Reconstruction

"Mining Large Grained Assets at the Architectural Level" on page 7 introduced architecture reconstruction as a strategy that can be used when the original architecture has been eroded or when a large set of interconnected components can potentially be inserted into a new architecture. The strategy of architecture reconstruction can be implemented through a number of different specific processes. Because of its importance for product line systems, a specific architecture reconstruction process with tool support from the Dali tool set is discussed in this section. We will outline the process and how it supports technical understanding and rehabilitation of assets. Further information about architecture reconstruction can also be obtained from O'Brien.[2]

Architecture reconstruction uses existing system and software artifacts to reconstruct the architecture of an implemented system. Relative to the activities for the mining of assets outlined in Section 4, architecture reconstruction requires a detailed technical understanding of assets. This is accomplished through an analysis of the system using tool support to collect data, and build and aggregate various levels of abstraction to obtain an architecture representation of that system. There is extensive exploration of alternative hypotheses using a variety of alternative visual techniques. The mechanisms used in architecture reconstruction to obtain a detailed system understanding are discussed in Section 5.2.

The reconstructed architecture may next require rehabilitation through the strategies of architecture transformation or architecture based development. For example, architecture transformation may be appropriate if the reconstructed architecture is viable but in need of some modifications to meet new quality attribute requirements.

In some cases it may not be possible to generate a reasonable architectural representation. For example, a system may have no inherent architectural components as a result of inconsistent architectural decision making or inconsistent application of architectural decisions.

5.1 Role of Reconstruction

Architecture reconstruction typically results in an architectural representation that can:

- be used for documenting the existing architecture
- be used to check the conformance of the as-implemented architecture to the as-designed architecture

[2] O'Brien, L. & Kazman, R. *Architecture Extraction Guidelines*. Pittsburgh, PA.: Software Engineering Institute, Carnegie Mellon University, forthcoming.

- serve as a starting point for reengineering the system to a new desired architecture, through the strategy of architecture transformation
- be used to identify components for establishing a product line approach

If an organization does not have up-to-date documentation for its existing system, it is often possible to reconstruct the architecture of the system to provide current documentation. Using automated support the source units that make up architectural components and the links between them serve as the basis for building the documentation.

In many cases the *as-implemented* architecture of a system will have drifted from *the as-designed* architecture. In such cases reconstructing the software architecture assists in checking conformance of the *as-implemented* architecture to the *as-designed* architecture.

In other cases an organization may want to update and add functionality to the system. The *as-implemented* architecture is then reconstructed and used as the basis for transformation to a new *as-desired* architecture.

When introducing a product line approach it is often beneficial to use existing architectural components in the product line. In these cases, architecture reconstruction can help to identify common components that can become core assets in the new product line architecture.

5.2 Dali Workbench

Architecture reconstruction is a complex set of tasks with a wide range of activities. Tool support using multiple tools for different tasks is often required. The diversity of tasks and different types of required tools has led to the concept of a workbench for supporting these activities. A *workbench* provides a lightweight integration framework so incorporating new tools does not have an unnecessary impact on existing tools or data. Moreover, the framework needs to be open, allowing for the easy integration of new tools. An example of such a workbench is Dali [Kazman 99].

Software architecture reconstruction using Dali has the following phases:

- view extraction
- database construction
- view fusion
- architecture reconstruction
- architecture analysis

The phases are highly iterative. Figure 2 illustrates the tasks of architecture reconstruction as supported by Dali. An outline of each phase is provided in the following sections.

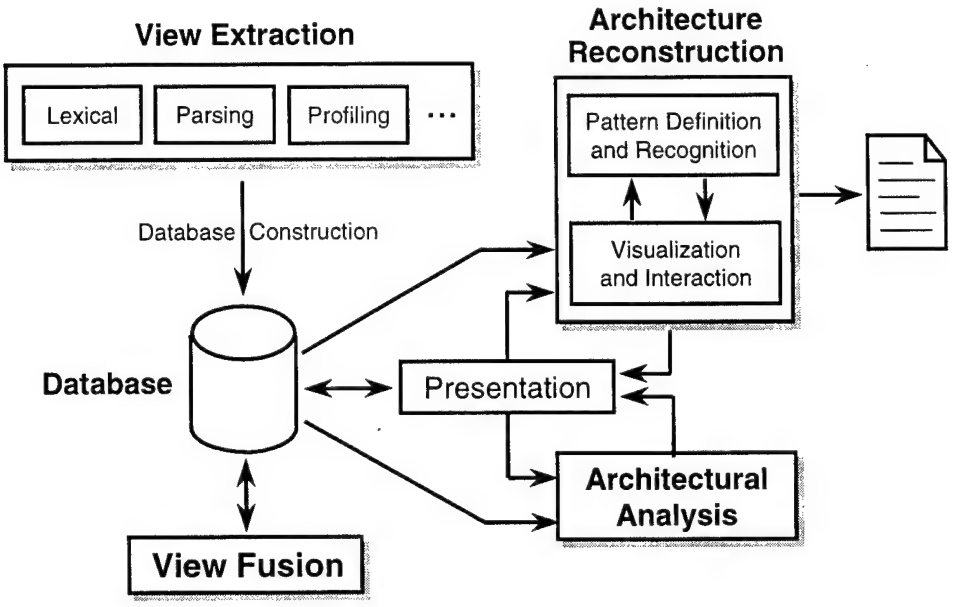

Figure 2: Overview of the Dali Workbench and the Various Phases

View Extraction

View extraction analyzes existing design and implementation artifacts and constructs an extracted view of the system. A view comprises a set of elements (files, functions, variables, etc.) and a set of relationships (calls, contains, includes, etc.) among these elements. Views can be either static or dynamic. Static views are obtained by observing only the artifacts of the system, while dynamic views are obtained by observing the execution of the system. In many cases static and dynamic views can be fused.

There are several different types of tools that support view extraction. Lexically based tools and parsers extract static views of the source code. Profiling, code coverage analysis and instrumentation generate dynamic views of the system. Tools that are designed to analyze design models, "makefiles" and executables can also be used to support extraction.

Database Construction

The extracted views are converted into the Dali format and stored in a relational database. SQL and Perl assist in the manipulation of the architectural views in the relational database and in providing reasoning capacities about these views.

View Fusion

View fusion is the process of defining and manipulating extracted views to create *fused* views. It reconciles and establishes connections between views. Different views provide complementary information. For example, if we have two views of a "function calls

function" relation in a system with dynamic binding, one view generated by carrying out a static analysis and the other obtained by generating a dynamic view, we can combine the information in both views. Neither of the two views may be complete in themselves, but the fused view would give us more complete information about the system.

Architecture Reconstruction

Architecture reconstruction has two main types of activities:

1. *visualization and interaction* enables the user to interactively visualize, explore and manipulate views through hierarchically decomposed graphs
2. *pattern definition and recognition* enables the definition and recognition of architectural patterns. Dali's architecture reconstruction facilities allow a user to construct more abstract views of a software system from more detailed ones by identifying aggregations of elements. Patterns are defined in Dali using a combination of SQL and Perl, and these patterns are stored in files. Users can selectively apply various patterns.

Architecture Analysis

During architecture analysis the conformance of the *as-reconstructed* architecture is evaluated relative to the *as-designed* architecture. As a starting point it is necessary to have the *as-designed* architecture from either the system documentation or the original developers. A representation of this architecture is entered into the Dali workbench using the visual presentation capabilities of *Rigi*. The representations of the two architectures in the Dali tool are exported from Dali to the *RMTool* [Murphy 95] where the conformance between the two architectures is analyzed.

5.3 Related Architecture Reconstruction Efforts

Several other efforts have been established in architecture analysis and reconstruction. Bowman, et al. [Bowman 99] outline a similar method for extracting architectural documentation from the code of an implemented system. In their example they used the Linux system. They analyze source code using *cfx* to obtain symbol information from the code and generate a set of relations between the symbols. They manually create a tree-structured decomposition of the Linux system into subsystems and assign the source files to these subsystems. They used the *grok* tool to determine relations between those subsystems and the *ledit* visualization tool is used to visualize the extracted system structure. Refinement of the resultant structure was carried out by moving source files between subsystems.

Harris, et al. [Harris 95] outline a framework for architecture reconstruction using a combined bottom-up and a top-down approach. The framework consists of three components: the architecture representation, the source code recognition engine and supporting library of recognition queries, and a "Bird's Eye" program overview capability. The bottom-up analysis uses the bird's eye view to display file structure and file components of a system and to reorganize information into more meaningful clusters. The top-down

analysis uses particular architectural styles to define components that should be found in the software. Recognition queries are then run to determine if the expected components exist.

6 Summary and Conclusions

Mining of existing assets represents a critical activity in implementing a software product line. However, there are significant risks involved because of the poor quality of many existing systems.

When assets are mined for product lines, additional factors need to be considered. Product lines have the additional requirement that mined assets fit into a software architecture. For product lines, the ideal mined assets are strategic, large-grained assets to meet the needs of schedule, cost, and quality. In addition, quality attributes such as maintainability and suitability over time assume greater importance because they affect the asset's total cost over a range of products. While the general field of reengineering has established a certain amount of maturity, there has not been a codification of strategies and processes for the mining of assets for product lines.

We have outlined four basic steps that are required to successfully mine assets:

1. preliminary information gathering which gathers baseline data about the system and available resources
2. decision-making on whether to mine assets and the type of overall strategy to use; a decision model, OAR offers a mechanism for making decisions on overall reengineering strategy
3. technical understanding of existing software, which includes the activities of detailed data collection, knowledge management, and exploration
4. rehabilitation of assets, through such strategies as architecture transformation at the architecture level, or through various wrapping approaches for components

Because of its importance as a strategy for enabling the mining of assets for product lines, we discussed a specific process for architecture reconstruction, as supported by an automated tool set we call the Dali workbench.

References

Bass 97 Bass, L. Clements, P.; & Kazman, R. *Software Architecture in Practice*. New York: Addison-Wesley, 1997.

Bass 99 Bass, L. & Kazman, R. *Architecture-Based Development* (CMU/SEI-99-TR-007, ADA366100). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tr007/99tr007abstract.html>.

Bergey 99 Bergey, J.; Smith, D.; Weiderman, N.; & Woods, S.N. *Options Analysis for Reengineering (OAR): Issues and Conceptual Approach.* (CMU/SEI-99-TN-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, October, 1999. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tn014/99tn014abstract.html>.

Bowman 99 Bowman, T.; Holt, R.C.; & Brewster, N.V. "Linux as a Case Study: Its Extracted Software Architecture." 555-563, *Proceedings of the 21st International Conference on Software Engineering (ICSE '99),* Los Angeles CA, May 16-22, 1999. New York: ACM Press Books, 1999.

Clements 99 Clements, P., et al. *A Framework for Software Product Line Practice - Version 2.0* [online]. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, July 1999. Available WWW: <URL: http://www.sei.cmu.edu/plp/framework.html>.

Comella-Dorda 00 Comella-Dorda, S.; Wallnau, K.; Seacord, R.C.; & Robert, J. *A Survey of Legacy System Modernization Approaches* (CMU/SEI-2000-TN-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, April 2000. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html>.

Harris 95 Harris, D.R.; Reubenstein, H.B.; & Yeh, A.S. "Reverse Engineering to the Architectural Level." 186-195, *17th International Conference on Software Engineering*, Seattle, WA, April 23-30, 1995. New York: ACM Press Books, 1995.

Kazman 99 Kazman, R. & Carriere, S. J. "Playing Detective: Reconstructing Software Architecture from Available Evidence." *Automated Software Engineering 6*, 2 (April 1999): 107-138

Muller 00 Muller, Hausi; Jahnke, Jens; Smith, Dennis; Storey, Margaret-Anne; Tilley, Scott; & Wong, Kenny. "Reverse Engineering: A Roadmap." *Future of Software Engineering*. New York: ACM Press Books, 2000.

Murphy 95 Murphy, G.C.; Notkin, D.; & Sullivan, K. "Software Reflexion Models: Bridging the Gap between Source and High-Level Models." *SIGSOFT Software Engineering Notes 20*, 4 (October 1995): 18-28.

Tilley 98 Tilley, S.R. *A Reverse-Engineering Environment Framework* (CMU/SEI-98-TR-005, ADA343688). Pittsburgh PA: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/98.reports/98tr005/98tr005title.htm>.

Woods 99 Woods, S.; Carriere, S.J.; & Kazman, R. "A Semantic Foundation for Architectural Reengineering and Interchange." 391-398, *Proceedings of the International Conference on Software Maintenance (ICSM-99)*. Oxford, England, August 30-September 3, 1999. Los Alamitos, CA: IEEE Computer Society, 1999.

Appendix: Definition of Key Concepts

We derive our definitions of the key concepts of software product lines and software architectures from the ongoing work of the SEI Software Product Line Systems Program.

Clements defines a software product line as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission [Clements 99]. Substantial economies can be achieved when the systems in a software product line are developed from a common set of core assets, in contrast to being developed one at a time in separate efforts. Using common assets, a new product is formed by taking applicable components from the asset base, tailoring them as necessary through pre-planned variation mechanisms such as parameterization, adding any new components that may be necessary, and assembling the collection under the umbrella of a common, product line-wide architecture. Building a new product (system) becomes more a matter of generation than creation; the predominant activity is integration rather than programming.

Software architecture forms the backbone for building successful software-intensive systems. A system's quality attributes are largely permitted or precluded by its architecture. Architecture represents a capitalized investment, an abstract reusable model that can be transferred from one system to the next. Software architecture forms one of the key reusable assets that form the basis of a software product line. Different products in the product line usually share the same architecture, or are built using prescribed variations of a common architecture. Bass defines a software architecture as the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [Bass 97].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 2000	Final

4. TITLE AND SUBTITLE
Mining Existing Assets for Software Product Lines

5. FUNDING NUMBERS
C — F19628-95-C-0003

6. AUTHOR(S)
John Bergey, Liam O'Brien, Dennis Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2000-TN-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Mining of existing assets offers an organization the potential to leverage all, or part, of its cumulative system investments, and thus represents a critical practice area in implementing a software product line. However, there are significant risks in achieving success because of the poorly documented and maintained state of many existing systems and the fact that many systems were initially developed for different paradigms than current distributed, Web-oriented, object-oriented approaches.

Four basic steps are required to successfully mine assets: 1) preliminary information gathering, 2) making decisions on whether to mine assets and which type of overall strategy to use, 3) obtaining detailed technical understanding of existing software assets, and 4) rehabilitation of assets.

This note outlines basic considerations for each of these steps. It outlines typical information to collect before an analysis. It then outlines a model for making decisions on mining legacy assets, and discusses the technical understanding of assets and the rehabilitation of assets.

Because of its importance as a strategy for product lines, architecture reconstruction is discussed, as it is supported by an automated tool set known as the Dali workbench.

14. SUBJECT TERMS
product lines, mining software assets, architecture reconstruction, reverse engineering

15. NUMBER OF PAGES
21 pp.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL